Kauno technologijos universitetas

Informatikos fakultetas

Interpoliavimas, aproksimavimas

P170B115 Skaitiniai metodai ir algoritmai

Trečias laboratorinis darbas

Projekto autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Vadovas
Andrius Kriščiūnas

Kaunas, 2020

Turinys

Įvadas

Užduoties variantas – 15.

1 užduotis. Interpoliavimas daugianariu

Duotas daugianaris

$$e^{-x^2} \cdot \cos(x^2) \cdot (x-3) \cdot (x^2+3);\ -3 \leq x \leq 3 \quad (1)$$

Pateikti interpoliacinės funkcijos išraišką naudojant vienanarių bazinę funkciją kai taškai pasiskirstę tolygiai ir kai taškai apskaičiuojami naudojant Čiobyševo abscises.

2 užduotis. Interpoliavimas daugianariu ir splainu per duotus taškus

Sudaryti interpoliuojančią kreivę Vengrijos 2006 m. temperatūrai per 12 mėnesių atvaizduoti vienanarių metodu ir naudojant globalųjį splainą.

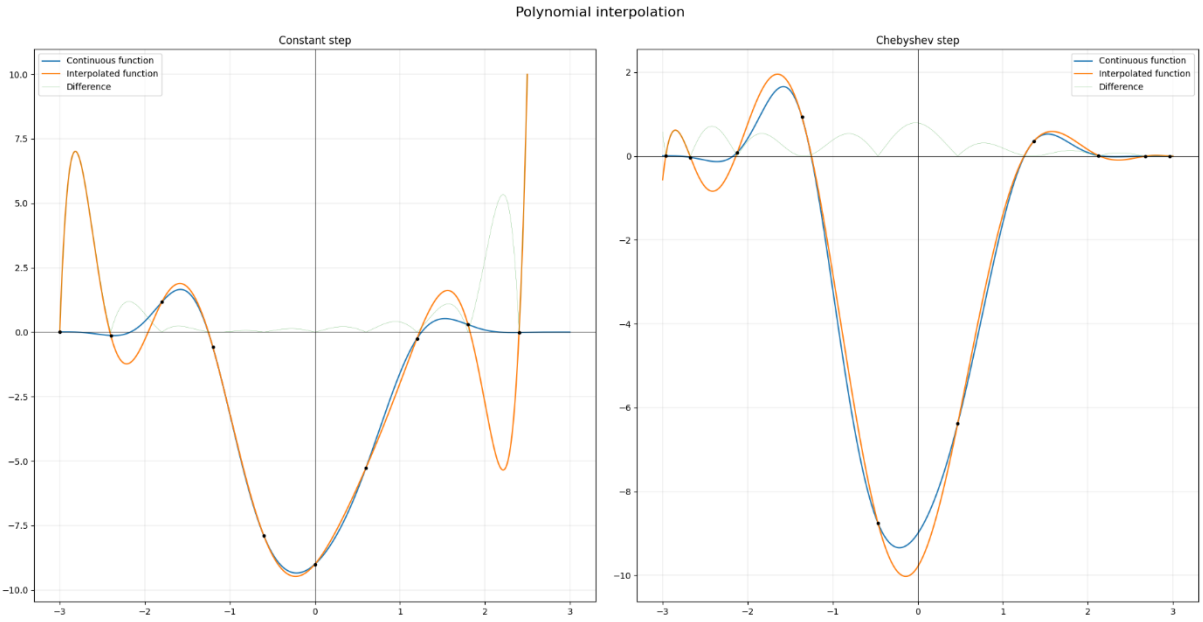
3 užduotis. Parametrinis interpoliavimas

Naudojant globalųjį splainą suformuoti Vengrijos kontūrą. Pasirinkti 10, 20, 50, 100 interpoliavimo taškų.

4 užduotis. Aproksimavimas

Pagal Vengrijos 2006 m. temperatūras sudaryti aproksimuojančią kreivę 12 mėnesių vidutinėms temperatūroms atvaizduoti naudojant antros, trečios, ketvirtos ir penktos eilės daugianarius.

Pirma užduotis

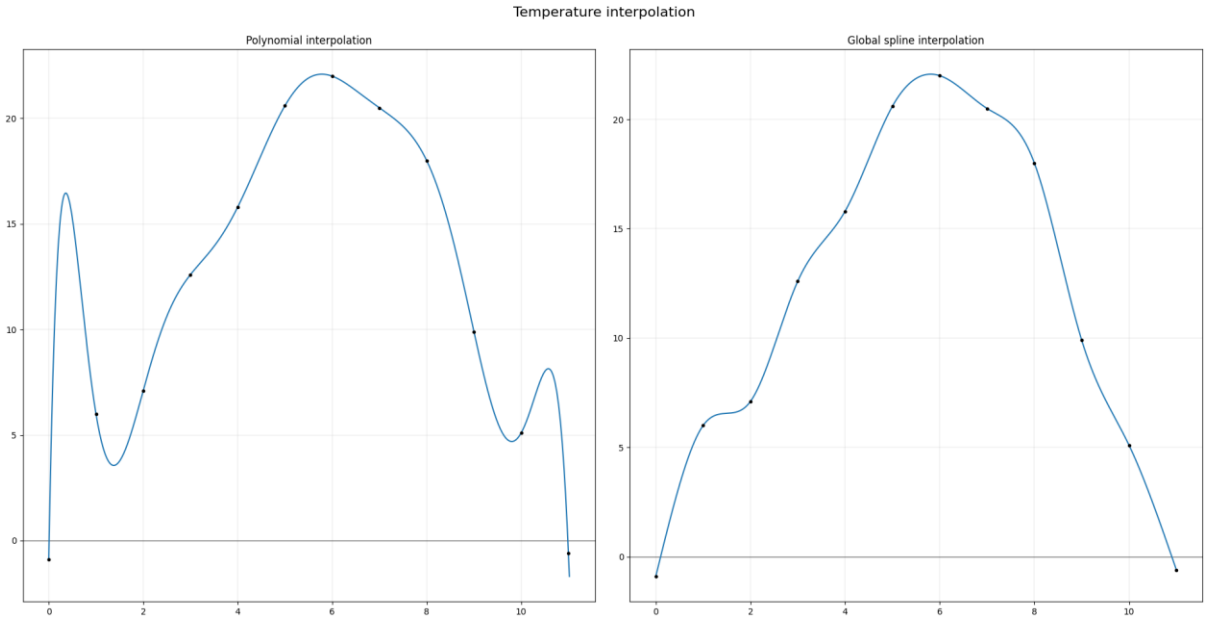


1 pav. Interpoliavimas daugianariu

1 pav. kairėje pavaizduotas tolygiu žingsniu pasiskirstę taškai, dešinėje – apskaičiuoti naudojant Čiobyševo abscises.

Interpoliuojant su tolygiai pasiskirsčiusiais taškais pradžioje ir gale gaunami labai stiprūs svyravimai. Pasinaudojus Čiobyševo abscisių metodu, svyravimai galuose yra žymiai sumažinami.

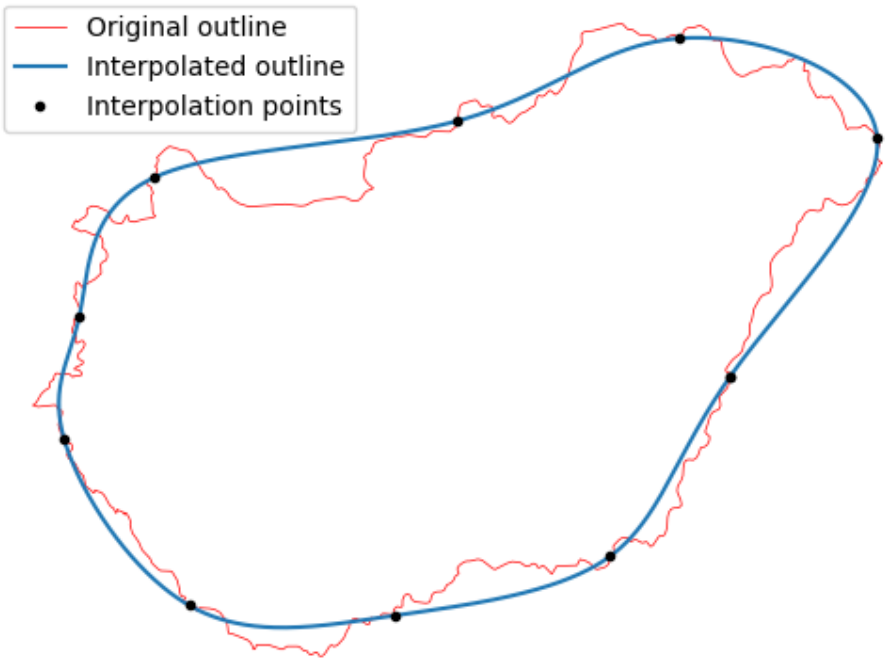
Antra užduotis



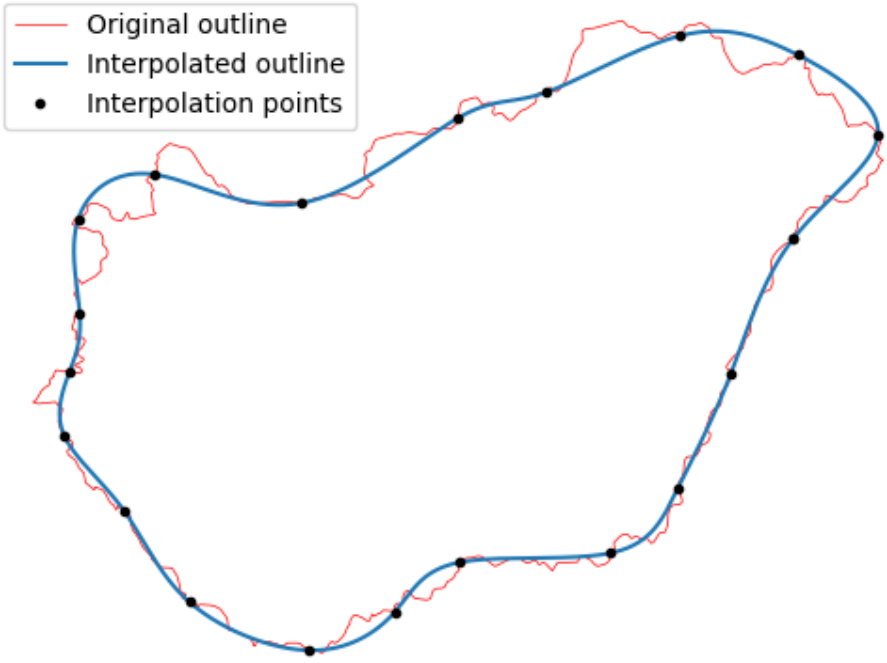
2 pav. Interpoliavimas daugianariu (kairėje) ir splainu (dešinėje)

Kaip ir 1 pav., 2 pav. aiškiai matosi stiprūs svyravimai interpoliavimo pradžioje ir gale. Tuo tarpu naudojant globalųjį splainą visiškai panaikinami tokie svyravimai.

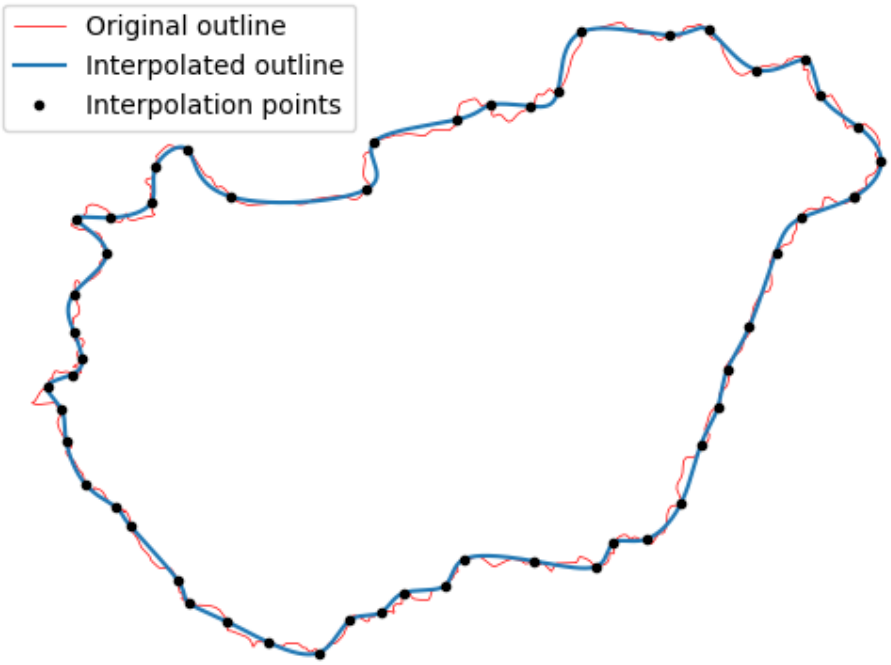
Trečia užduotis



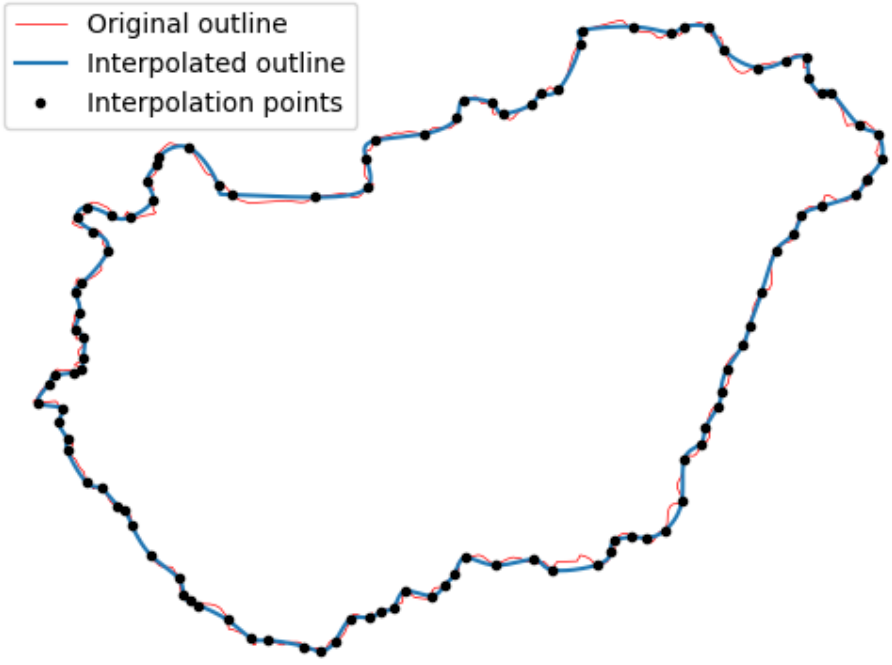
3 pav. Vengrijos kontūras naudojant 10 interpoliavimo taškų



4 pav. Vengrijos kontūras naudojant 20 interpoliavimo taškų



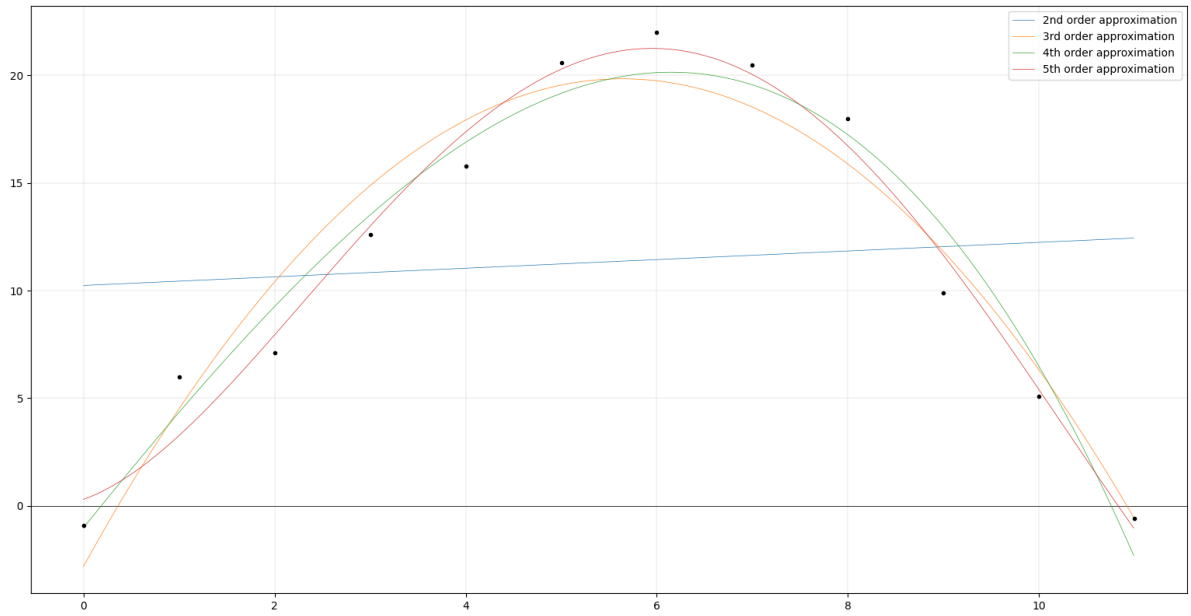
5 pav. Vengrijos kontūras naudojant 50 interpoliavimo taškų



6 pav. Vengrijos kontūras naudojant 100 interpoliavimo taškų

Paveiksluose 3 – 6 pavaizduoti gauti rezultatai su skirtingu skaičiumi interpoliavimo taškų. Kitaip nei antroje užduotyje, čia naudojamas periodinis splainas, kad kreivės pradžia ir pabaiga būtų glotni.

Ketvirta užduotis



7 pav. Vengrijos temperatūros aproksimacija

7 pav. atvaizduota Vengrijos 2006 m. temperatūrų aproksimacijos naudojant antros, trečios, ketvirtos ir penktos eilės daugianarius.

Gauti daugianariai (koeficientai suapvalinti iki dviejų skaičių po kablelio dėl aiškumo):

$$f_2(x) = 10.24 + 0.2x$$

$$f_3(x) = -2.82 + 8.04x - 0.71x^2$$

$$f_4(x) = -1.02 + 5.51x - 0.11x^2 - 0.04x^3$$

$$f_5(x) = 0.3 + 1.59x + 1.66x^2 - 0.29x^3 + 0.01x^4$$

Programinis kodas

L3_1.py

```
import numpy as np
import matplotlib.pyplot as plt


FROM = -3
TO = 3
POINTS = 10
DETAILED_STEP = 0.001


def f(x):
    return np.exp(-x ** 2) * np.cos(x ** 2) * (x - 3) * (x ** 2 + 3)


def fi(a, x):
    y = 0
    power = 0

    for i in a:
        y += i * x ** power
        power += 1

    return y


def x_matrix(x_val):
    matrix = []

    for i in x_val:
        x_row = []

        for j in range(POINTS):
            x_row.append(i ** j)

        matrix.append(x_row)

    return matrix


def chebyshev_x():
    x = []

    for i in range(POINTS):
        x.append((TO - FROM) / 2 * np.cos(np.pi * (2 * i + 1) / (2 * POINTS)) + (TO + FROM) / 2)

    return np.array(x)


def plot_functions(x_values, axis, title):
    # Continuous function x_file and y_file values
    x_fun = np.arange(FROM, TO, DETAILED_STEP)
    y_fun = f(x_fun)

    y_values = f(x_values)
    a_values = np.linalg.solve(x_matrix(x_values), y_values)

    # Interpolated function x_file and y_file values
    x_interpolated = np.arange(FROM, TO, DETAILED_STEP)
    y_interpolated = fi(a_values, x_interpolated)

    # Remove large y_file values
    inter_filter = y_interpolated < 10
    y_interpolated = y_interpolated[inter_filter]
```

```
    x_interpolated = x_interpolated[:len(y_interpolated)]

    # Calculating function differences
    x_diff = x_fun[:len(x_interpolated)]
    y_diff = abs(y_interpolated - y_fun[:len(y_interpolated)])

    continuous_function, = axis.plot(x_fun, y_fun)
    interpolated_function, = axis.plot(x_interpolated, y_interpolated)
    difference_function, = axis.plot(x_diff, y_diff, linewidth=0.2)

    for i in range(len(x_values)):
        axis.plot(x_values[i], y_values[i], 'ko', markersize=3)

    axis.set_title(title)
    axis.axvline(linewidth=0.5, color='k')
    axis.axhline(linewidth=0.5, color='k')
    axis.grid(linewidth=0.2)
    axis.legend(
        [continuous_function, interpolated_function, difference_function],
        ['Continuous function', 'Interpolated function', 'Difference'])


fig, axs = plt.subplots(1, 2, constrained_layout=True)
fig.suptitle('Polynomial interpolation', fontsize=16)

plot_functions(np.arange(FROM, TO, (TO - FROM) / POINTS), axs[0], 'Constant step')
plot_functions(chebyshev_x(), axs[1], 'Chebyshev step')

plt.show()
```

L3_2.py

```
import matplotlib.pyplot as plt
import numpy as np


TEMPERATURE = [-0.9, 6.0, 7.1, 12.6, 15.8, 20.6, 22.0, 20.5, 18.0, 9.9, 5.1, -0.6]


fig, axs = plt.subplots(1, 2, constrained_layout=True)
fig.suptitle('Temperature interpolation', fontsize=16)


def plot_functions(x_arr, y_arr, axis, title):
    axis.set_title(title)
    axis.axhline(linewidth=0.5, color='k')
    axis.grid(linewidth=0.2)

    axis.plot(x_arr, y_arr)
    axis.plot(range(12), TEMPERATURE, 'ko', markersize=3)


# ----- Part 1 -----
def x_matrix():
    matrix = []

    for i in range(12):
        x_row = []

        for j in range(12):
            x_row.append(i ** j)

        matrix.append(x_row)

    return matrix


def interpolate(a, x_arr):
    y_arr = 0
    power = 0

    for i in a:
        y_arr += i * x_arr ** power
        power += 1

    return y_arr


a_values = np.linalg.solve(x_matrix(), TEMPERATURE)
x_interpolated = np.arange(0, 12, 0.01)
y_interpolated = interpolate(a_values, x_interpolated)

inter_filter = y_interpolated > min(TEMPERATURE) - 1
y_interpolated = y_interpolated[inter_filter]
x_interpolated = x_interpolated[:len(y_interpolated)]

plot_functions(x_interpolated, y_interpolated, axs[0], 'Polynomial interpolation')


# ----- Part 2 -----
def d_matrix():
    temp_len = len(TEMPERATURE)
    matrix = np.zeros((temp_len - 2, temp_len))

    for i in range(temp_len - 2):
        matrix[i][i] = 1 / 6
        matrix[i][i + 1] = 2 / 3
        matrix[i][i + 2] = 1 / 6

    # Remove first and last column to make the matrix square
    square = []
    for i in matrix:
        square.append(i[1:len(i) - 1])
```

```
    return square


def y_vector():
    vector = []

    for i in range(len(TEMPERATURE) - 2):
        vector.append(TEMPERATURE[i + 2] - 2 * TEMPERATURE[i + 1] + TEMPERATURE[i])

    return vector


def calc_y(f_prime0, f_prime1, y0, y1, step):
    return (f_prime0 * step ** 2 / 2 -
            f_prime0 * step ** 3 / 6 +
            f_prime1 * step ** 3 / 6 +
            (y1 - y0) * step -
            f_prime0 * step / 3 -
            f_prime1 * step / 6 +
            y0)


def global_spline_interpolation(f_primes):
    x_interpol = []
    y_interpol = []
    month = 0

    for step in np.arange(0, 1, 0.01):
        x_interpol.append(month + step)
        y_interpol.append(calc_y(0, f_primes[0], TEMPERATURE[0], TEMPERATURE[1], step))

    month += 1

    for i in range(len(f_primes) - 1):
        for step in np.arange(0, 1, 0.01):
            x_interpol.append(month + step)
            y_interpol.append(
                f_primes[i] * step ** 2 / 2 -
                f_primes[i] * step ** 3 / 6 +
                f_primes[i + 1] * step ** 3 / 6 +
                (TEMPERATURE[i + 1 + 1] - TEMPERATURE[i + 1]) * step -
                f_primes[i] * step / 3 -
                f_primes[i + 1] * step / 6 +
                TEMPERATURE[i + 1]
            )

        month += 1

    for step in np.arange(0, 1, 0.01):
        x_interpol.append(month + step)
        y_interpol.append(calc_y(f_primes[len(f_primes) - 1], 0, TEMPERATURE[10], TEMPERATURE[11],
step))

    return x_interpol, y_interpol


f_primes2 = np.linalg.solve(d_matrix(), y_vector())
x, y = global_spline_interpolation(f_primes2)

plot_functions(x, y, axs[1], 'Global spline interpolation')


plt.show()
```

L3_3.py

```
import numpy as np
import matplotlib.pyplot as plt


def read_file(point_num):
    x_arr = []
    y_arr = []

    file = open('country/coords.txt', 'r')
    lines = file.readlines()

    i = 1
    skip = int(len(lines) / point_num)

    for line in lines:
        if i % skip == 0:
            coords = line.split(' ')
            x_arr.append(int(coords[0]))
            y_arr.append(int(coords[1]))

        i += 1

    x_arr.append(x_arr[0])
    y_arr.append(y_arr[0])

    file.close()

    return x_arr, y_arr


def d_matrix(arr):
    arr_len = len(arr)
    matrix = np.zeros((arr_len, arr_len))

    for i in range(arr_len - 2):
        matrix[i][i] = 1 / 6
        matrix[i][i + 1] = 2 / 3
        matrix[i][i + 2] = 1 / 6

    matrix[arr_len - 2][0] = 1 / 3
    matrix[arr_len - 2][1] = 1 / 6
    matrix[arr_len - 2][arr_len - 2] = 1 / 6
    matrix[arr_len - 2][arr_len - 1] = 1 / 3

    matrix[arr_len - 1][0] = 1
    matrix[arr_len - 1][arr_len - 1] = -1

    return matrix


def y_vector(arr):
    n = len(arr)
    vector = []

    for i in range(n - 2):
        vector.append(arr[i] - 2 * arr[i + 1] + arr[i + 2])

    vector.append(arr[1] - arr[0] - arr[n - 1] + arr[n - 2])
    vector.append(0.0)

    return vector


def calc_y(f_prime0, f_prime1, y0, y1, step):
    return (f_prime0 * step ** 2 / 2 -
            f_prime0 * step ** 3 / 6 +
            f_prime1 * step ** 3 / 6 +
            (y1 - y0) * step -
            f_prime0 * step / 3 -
            f_prime1 * step / 6 +
            y0)
```

```
def global_spline_interpolation(arr, f_primes):
    interpol_values = []

    for i in range(len(arr) - 1):
        for step in np.arange(0, 1, 0.01):
            interpol_values.append(calc_y(
                f_primes[i],
                f_primes[i + 1],
                arr[i],
                arr[i + 1],
                step,
            ))

    return interpol_values


x_full, y_full = read_file(847)
full_outline, = plt.plot(x_full, y_full, 'r', linewidth=0.5)


x, y = read_file(100)
x_primes = np.linalg.solve(d_matrix(x), y_vector(x))
y_primes = np.linalg.solve(d_matrix(y), y_vector(y))
interpolated_x = global_spline_interpolation(x, x_primes)
interpolated_y = global_spline_interpolation(y, y_primes)

interpolated_outline, = plt.plot(interpolated_x, interpolated_y)
interpolation_points, = plt.plot(x, y, 'ko', markersize=3)

plt.legend(
    [full_outline, interpolated_outline, interpolation_points],
    ['Original outline', 'Interpolated outline', 'Interpolation points']
)

plt.axis('off')
plt.show()
```

L3_4.py

```
import numpy as np
import matplotlib.pyplot as plt


MONTHS = range(12)
TEMPERATURE = [-0.9, 6.0, 7.1, 12.6, 15.8, 20.6, 22.0, 20.5, 18.0, 9.9, 5.1, -0.6]


def g(num_approx):
    matrix = []

    for x in MONTHS:
        row = []

        for j in range(num_approx):
            row.append(x ** j)

        matrix.append(row)

    return matrix


def c(num_approx):
    g_matrix = g(num_approx)
    g_t = np.transpose(g_matrix)

    left = np.matmul(g_t, g_matrix)
    right = np.matmul(g_t, TEMPERATURE)

    return np.linalg.solve(left, right)


def approximate(coeff):
    x = np.arange(0, 11, 0.01)
    y = []

    for i in x:
        val = 0

        for p, j in enumerate(coeff):
            val += j * i ** p

        y.append(val)

    return x, y


def plot(num_approx):
    coeff = c(num_approx)
    print("------")
    print("Approximation order =", num_approx)
    print("c =", coeff)

    x_approx, y_approx = approximate(coeff)

    line, = plt.plot(x_approx, y_approx, linewidth=0.5)
    return line


plt.axhline(linewidth=0.5, color='k')
plt.grid(linewidth=0.2)
lines = []

for i in range(2, 6):
    line = plot(i)
    lines.append(line)

plt.plot(MONTHS, TEMPERATURE, 'ko', markersize=3)
plt.legend(lines, [
    "2nd order approximation",
    "3rd order approximation",
    "4th order approximation",
```

```
    "5th order approximation",
])
plt.show()
```